Pipelining High-Radix SRT Division Algorithms

Saurabh Upadhyay

Illinois Institute of Technology

Outline

- Introduction
- Division by digit recurrence and pipelining
- Use of different memory elements to pipeline the algorithm in static CMOS
- Use of pass-transistor logic
- Domino clocking techniques for the algorithm
- comparison and conclusion

Introduction

- SRT Division is very popular division algorithm used in current microprocessors.
- It is named for D. Sweeny, J. E. Robertson and K. D. Tocher.
- This dissertation shows different ways to pipeline SRT division algorithm.
- Simulation results are compared to find out the fastest possible implementation.

What current microprocessors use?

Processor	Division Algorithm	Connectivity
DEC 21164 Alpha AXP	SRT	Adder-Coupled
Hal Sparc64	SRT	Independent
HP PA7200	SRT	Independent
HP PA8000	SRT	Multiplier-accumulate-c/p
IBM RS/6000 Power2	Newton-Raphson	Integrated
Intel Pentium	SRT	Adder-coupled
Intel Pentium Pro	SRT	Independent
Mips R8000	Multiplicative	Integrated
Mips R10000	SRT	Multiplier-coupled
PowerPc 604	SRT	Integrated
PowerPc 620	SRT	Integrated
Sun SuperSparc	Goldschmidt	Multiplier-integrated
Sun UltraSparc	SRT	Independent

Division by Digit Recurrence

- $x = q \cdot d + rem$

Division by Digit Recurrence

- $x = q \cdot d + rem$
- Divisor is normalized, i.e. $\frac{1}{2} \leq d < 1$ for fractional division.

Division by Digit Recurrence

- $x = q \cdot d + rem$
- Divisor is normalized, i.e. $\frac{1}{2} \leq d < 1$ for fractional division.
- For normalized fractional divisor the quotient is in the range $0 < q < 2$

Division by Digit Recurrence

- $x = q \cdot d + rem$
- Divisor is normalized, i.e. $\frac{1}{2} \leq d < 1$ for fractional division.
- For normalized fractional divisor the quotient is in the range $0 < q < 2$
- $w_{i+1} = r \cdot w_i - q_{i+1} \cdot d$
  where $w_0 = x$

Division by Digit Recurrence

- $x = q \cdot d + rem$
- Divisor is normalized, i.e. $\frac{1}{2} \leq d < 1$ for fractional division.
- For normalized fractional divisor the quotient is in the range $0 < q < 2$
- $w_{i+1} = r \cdot w_i - q_{i+1} \cdot d$
  where $w_0 = x$
- Consists of n iterations, each iteration produces one digit of the quotient.

Division by Digit Recurrence (continued)

- q_{i+1} is selected such that w_{i+1} is bounded.

Division by Digit Recurrence (continued)

- q_{i+1} is selected such that w_{i+1} is bounded.
- $q_{i+1} = QST(r \cdot w_i, d) \ldots$ Quotient Selection Table

Division by Digit Recurrence (continued)

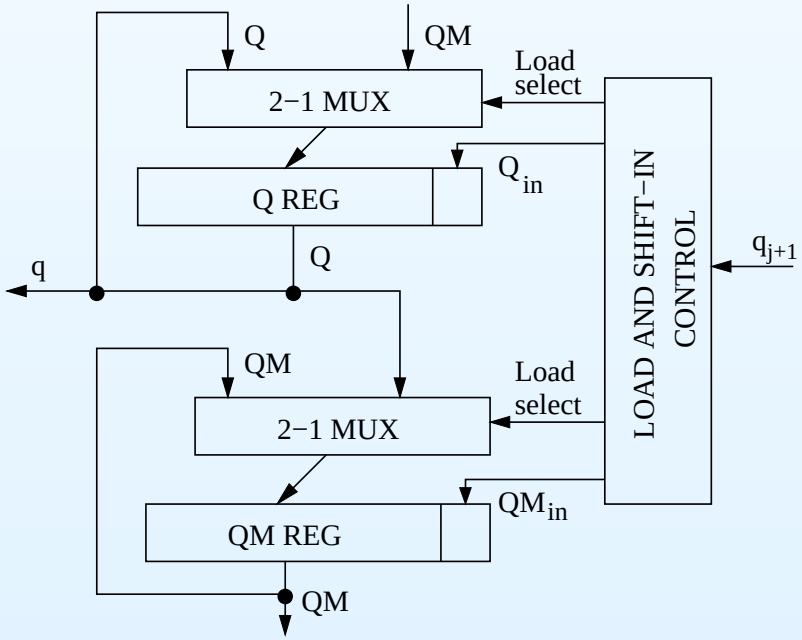
- q_{i+1} is selected such that w_{i+1} is bounded.
- $q_{i+1} = QST(r \cdot w_i, d) \ldots$ Quotient Selection Table
- $q = q[n] = q[0] + \sum_{i=1}^{n} q_i r^{-i}$

Division by Digit Recurrence (continued)

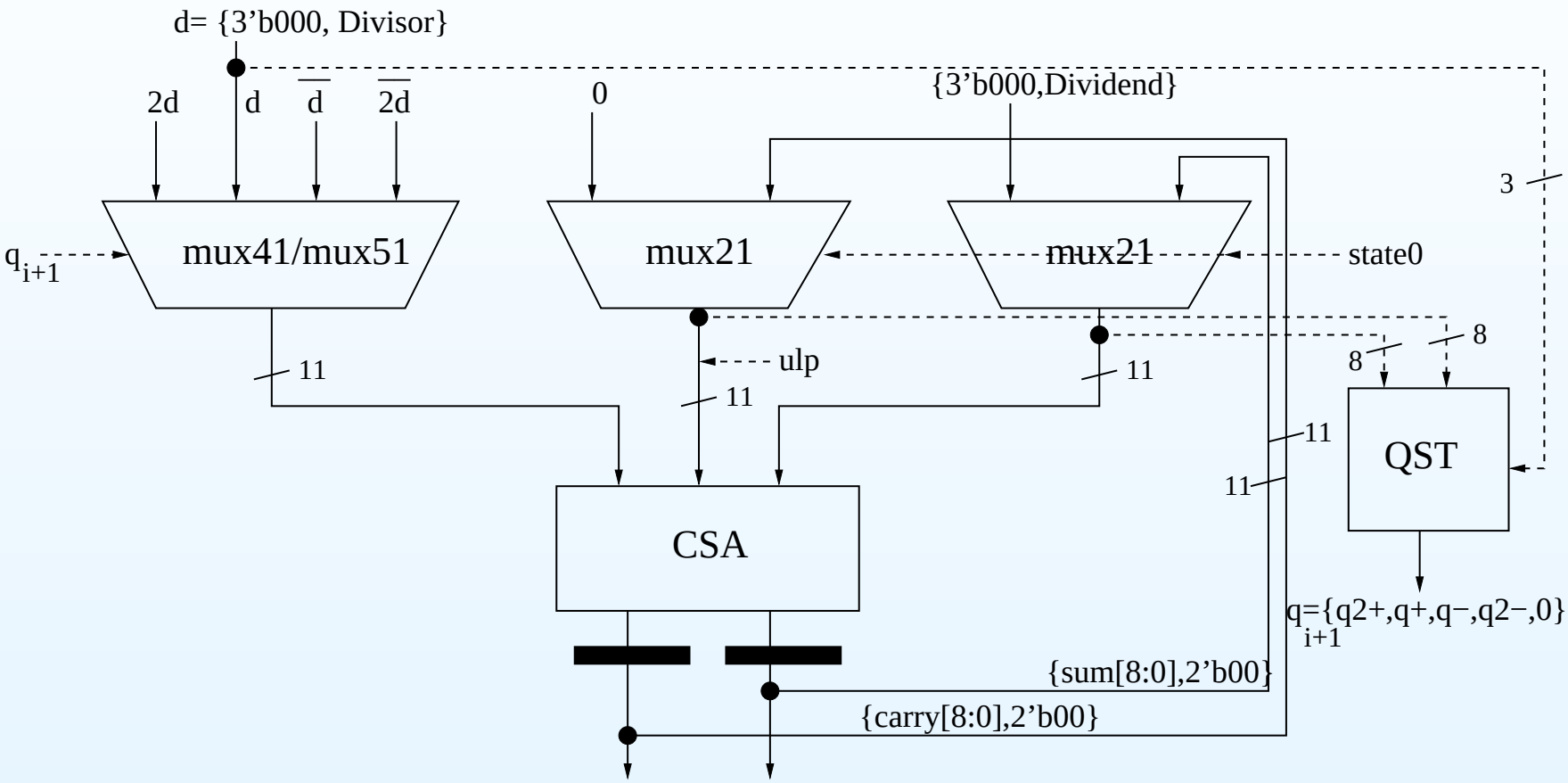
- q_{i+1} is selected such that w_{i+1} is bounded.
- $q_{i+1} = QST(r \cdot w_i, d) \ldots$ Quotient Selection Table
- $q = q[n] = q[0] + \sum_{i=1}^{n} q_i r^{-i}$
- Quotient is formed by concatenation of quotient bits.

Division by Digit Recurrence (continued)

- q_{i+1} is selected such that w_{i+1} is bounded.
- $q_{i+1} = QST(r \cdot w_i, d) \ldots$ Quotient Selection Table
- $q = q[n] = q[0] + \sum_{i=1}^{n} q_i r^{-i}$
- Quotient is formed by concatenation of quotient bits.
- On-the-fly conversion is used for redundant quotient digit set.



Basic SRT Division Algorithm

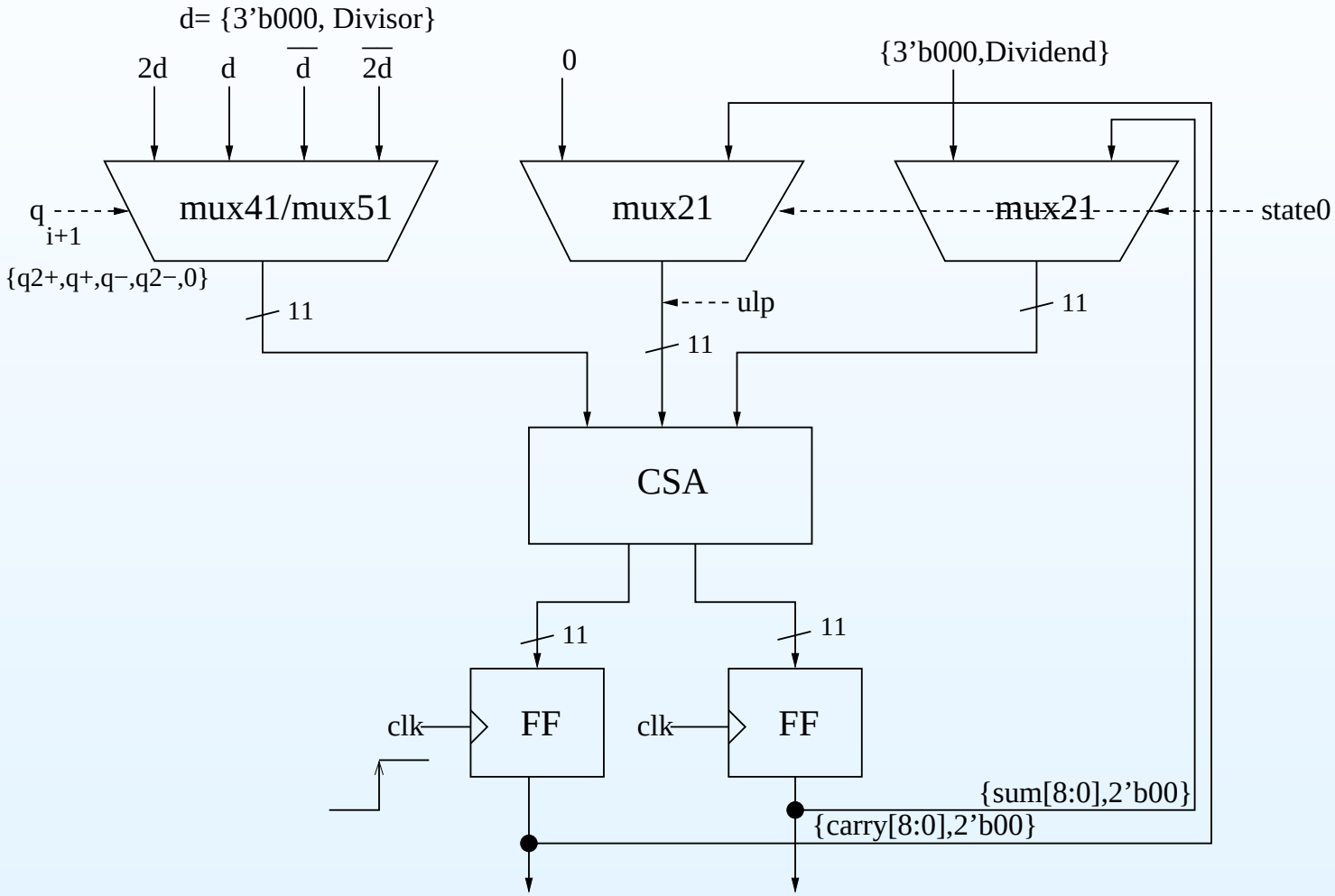


$$w_{i+1} = r \cdot w_i - q_{i+1} \cdot d$$

Pipelining

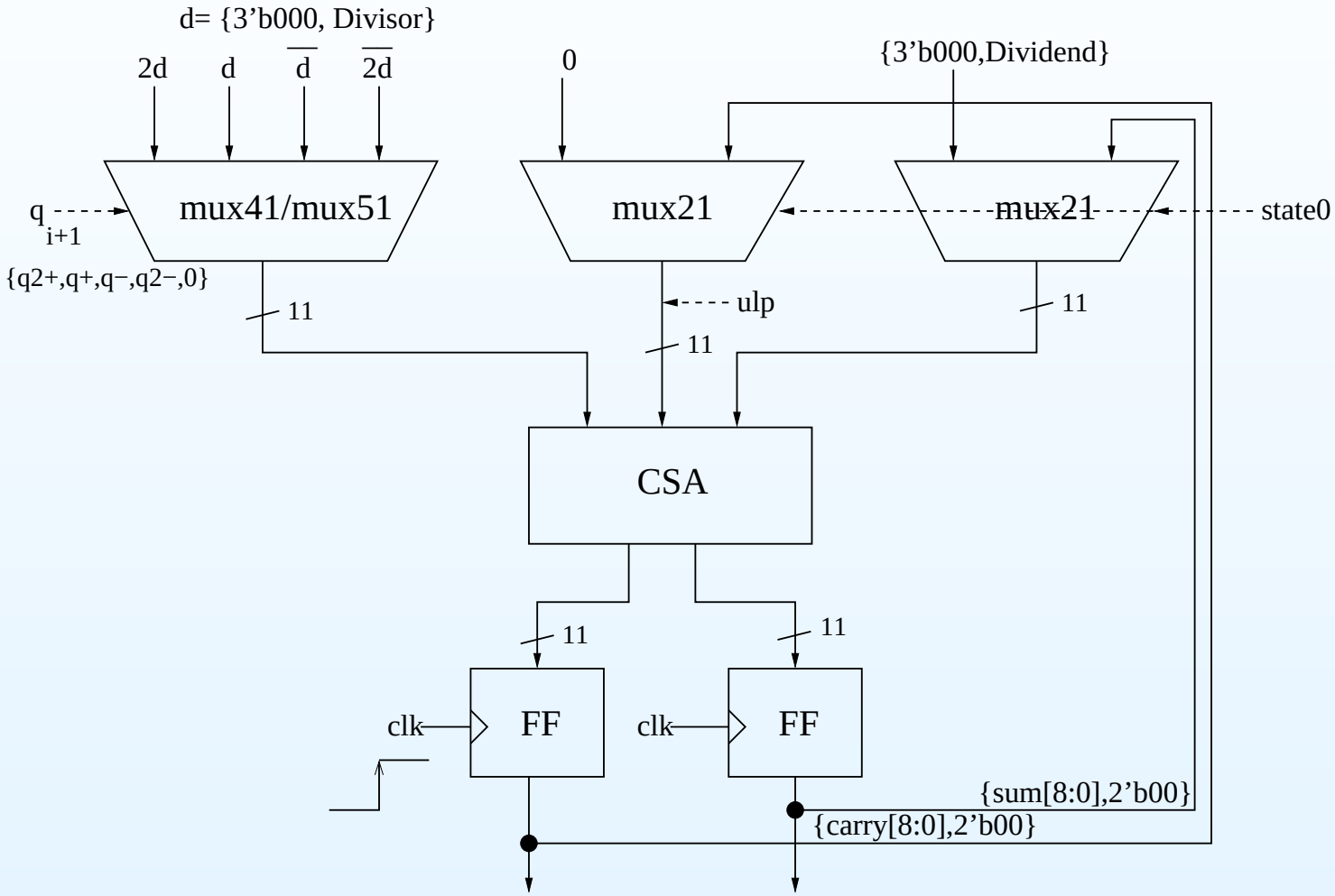
- Pipelining is a technique used to create clock cycle times for arithmetic data-paths.
- Usually, memory elements like flip-flops, transparent latches and pulsed latches are used to impose sequencing.
- Pipelining techniques also vary with different logic gate family.
- Intelligent pipelining technique can improve performance greatly, on the other hand poor pipelining introduces large sequencing overheads.

Pipelining with Flip-Flops



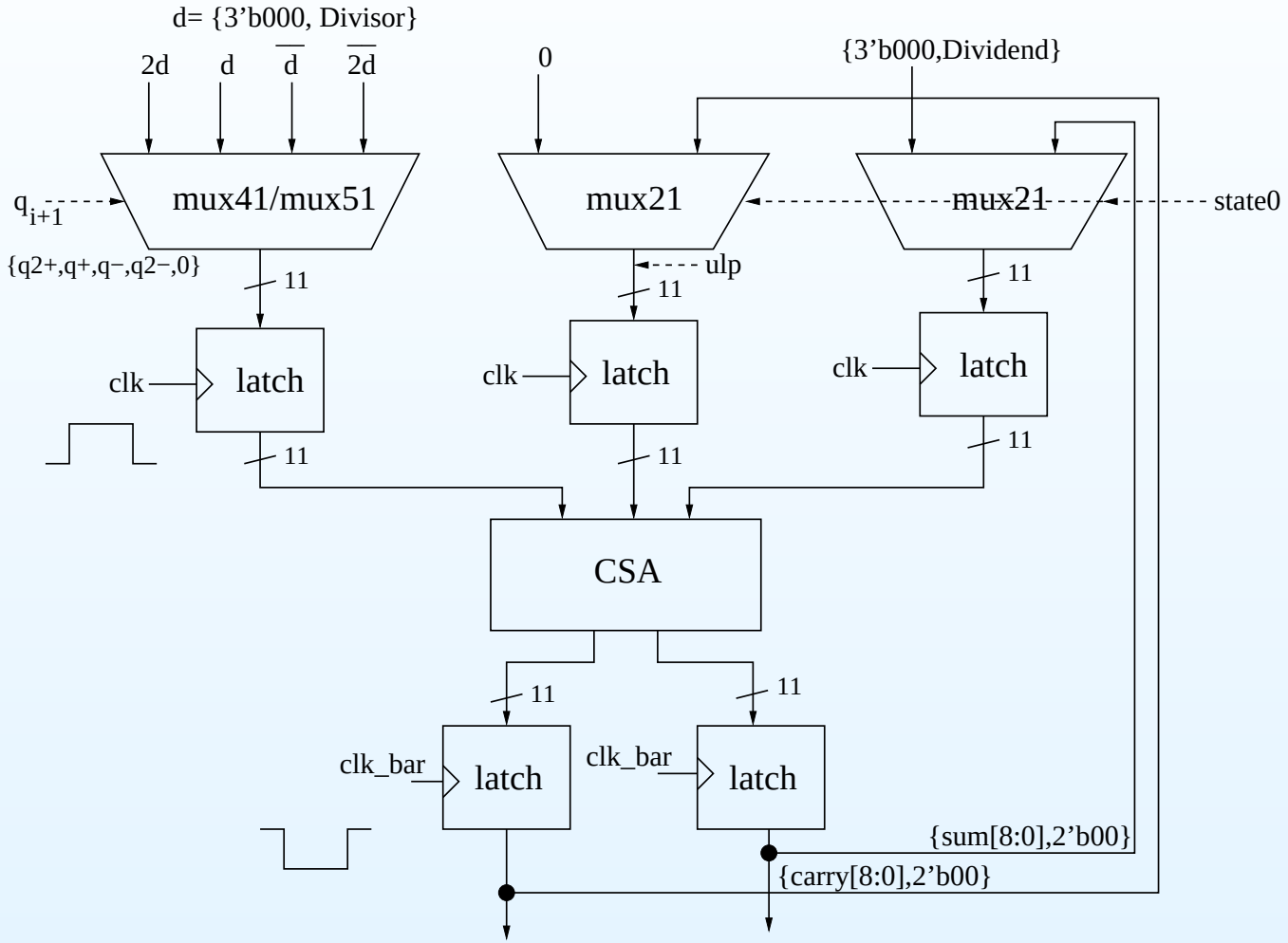
$$T_c = \Delta_{logic} + \Delta_{CQ} + \Delta_{DC} + t_{skew}$$

Pipelining with Flip-Flops



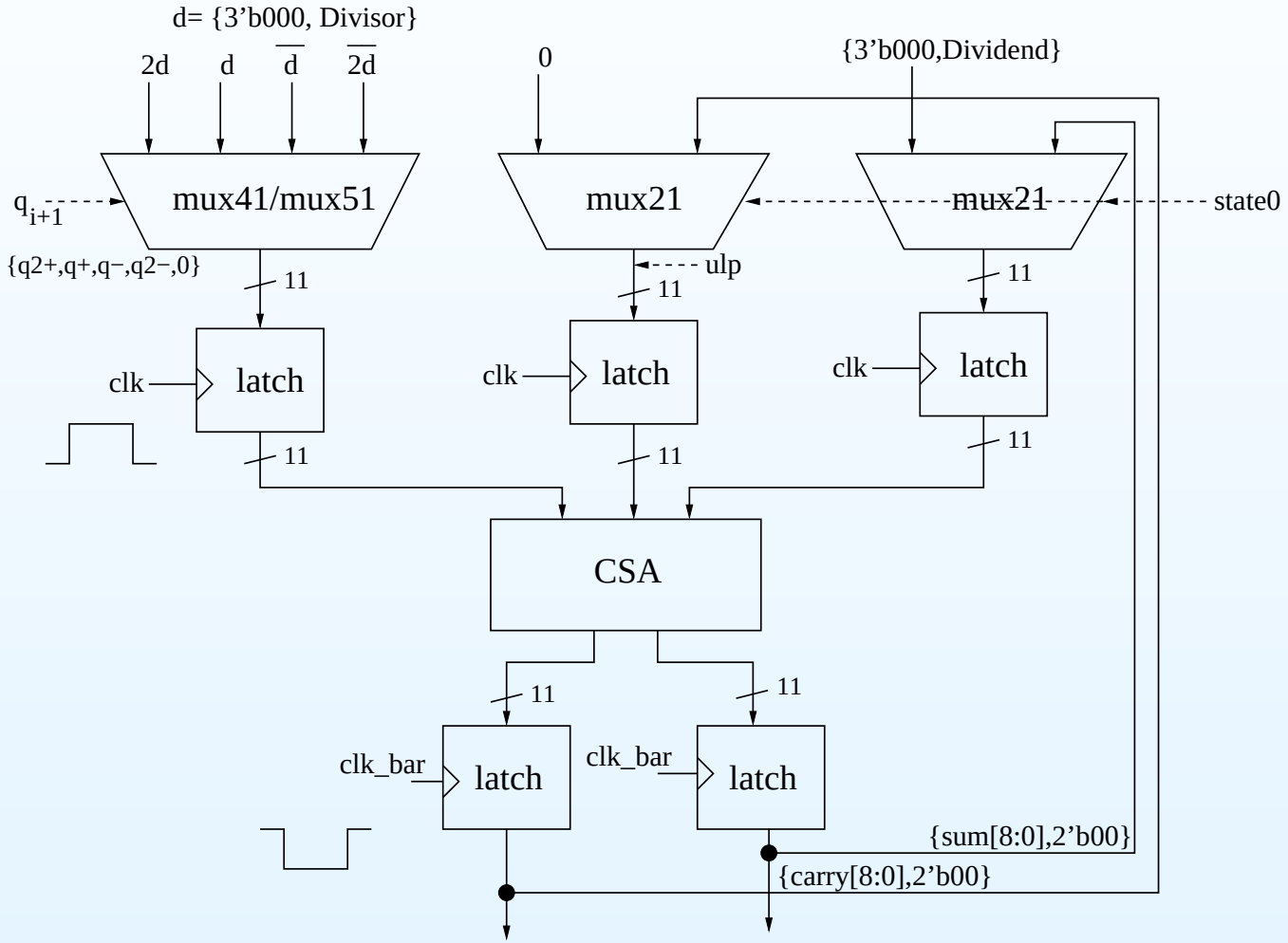
$$T_c = \Delta_{logic} + \Delta_{CQ} + \Delta_{DC} + t_{skew} = 2ns$$

Pipelining with Latches



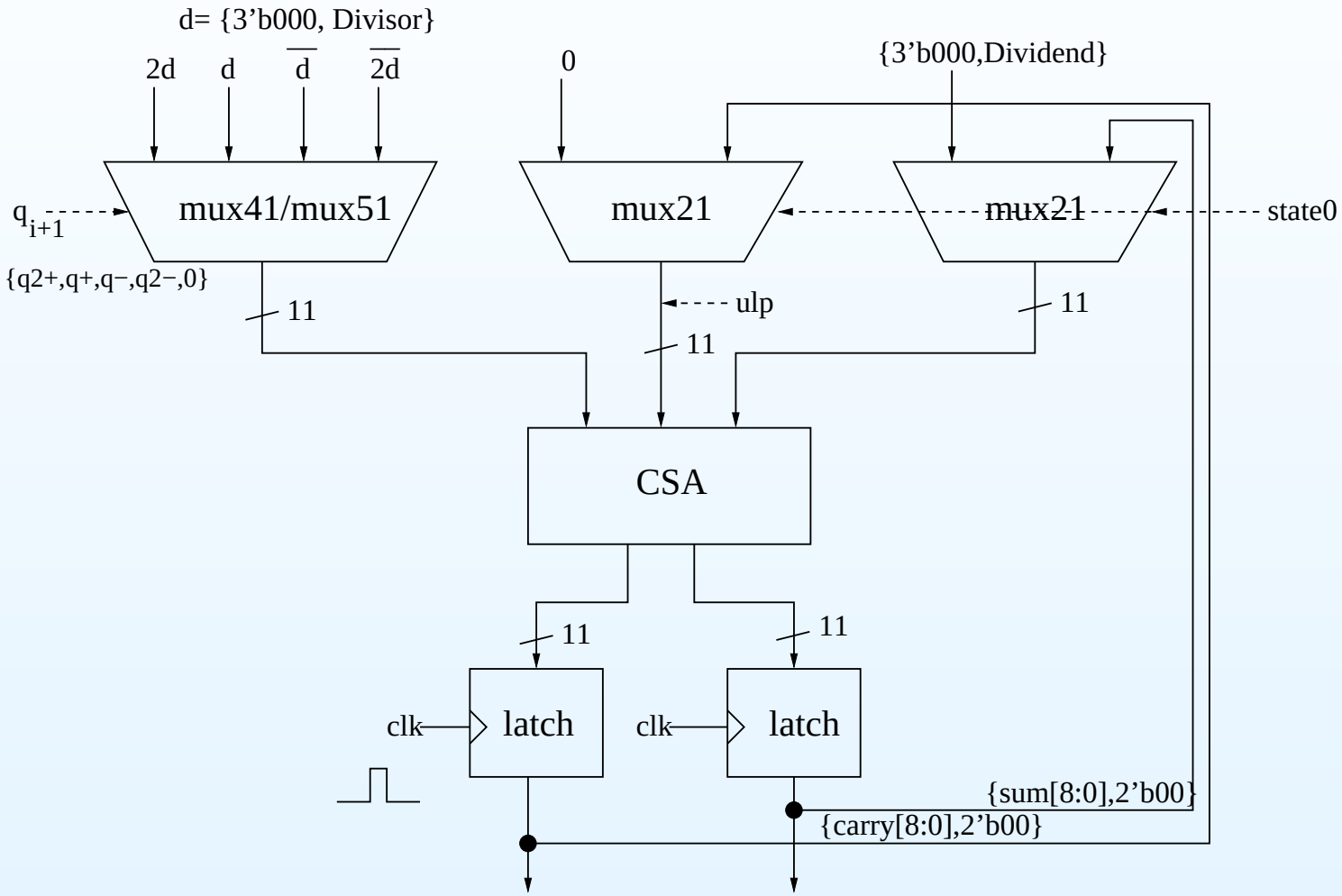
$$T_c = \Delta_{logic} + 2 \cdot \Delta_{DQ}$$

Pipelining with Latches



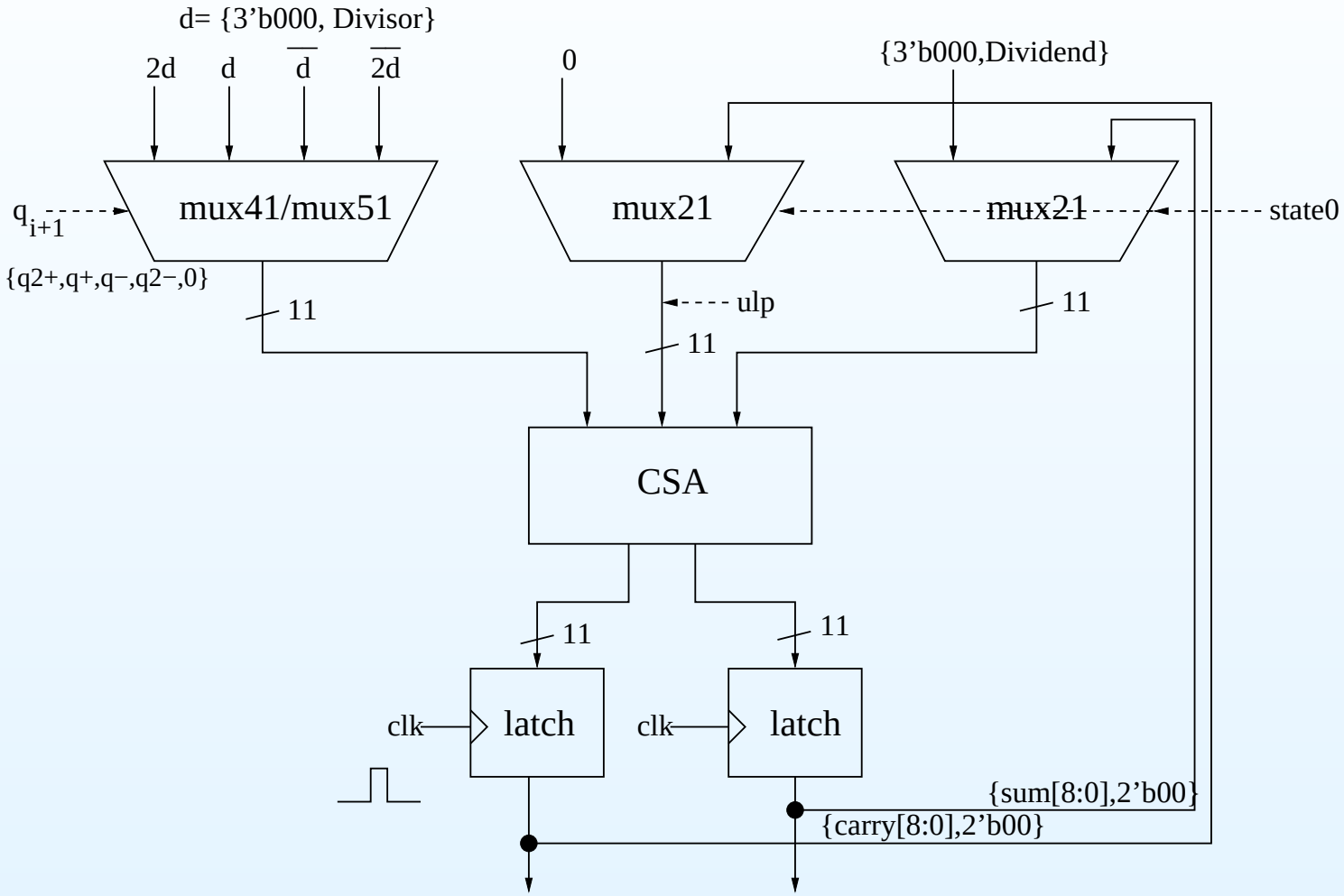
$$T_c = \Delta_{logic} + 2 \cdot \Delta_{DQ} = 1.8ns$$

Pipelining with Pulsed Latches



$$T_c = \Delta_{logic} + \Delta_{DQ} + max(0, \Delta_{DC} + t_{skew} - t_{pw})$$

Pipelining with Pulsed Latches



$$T_c = \Delta_{logic} + \Delta_{DQ} + max(0, \Delta_{DC} + t_{skew} - t_{pw}) = 1.6ns$$

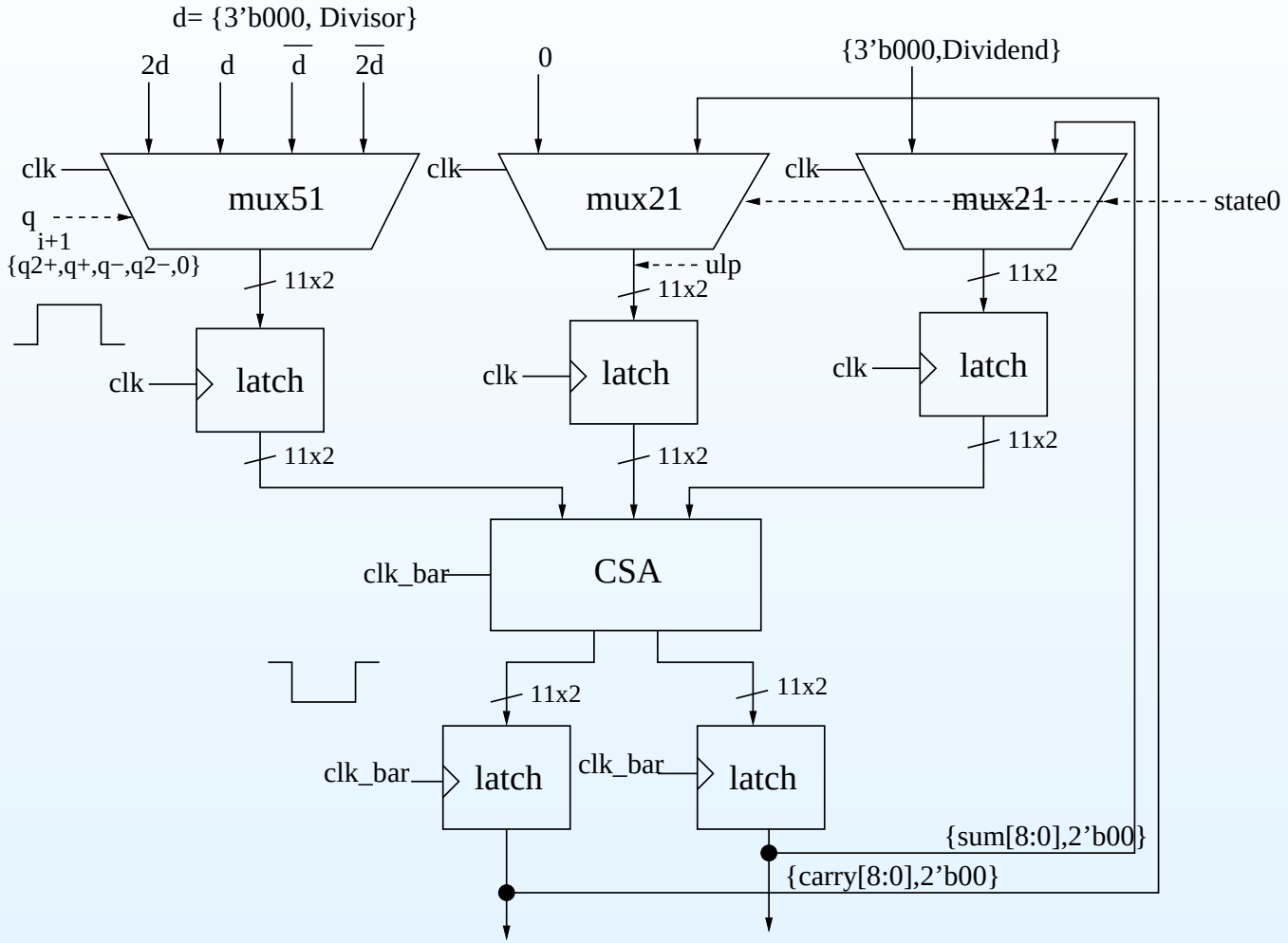
Use of Pass Transistor Logic

- Static multiplexors are replaced with transmission gate multiplexors, a well-known application of pass transistor logic.
- The algorithm now needs a 5-1 multiplexor instead of a 4-1 multiplexor.
- The time periods are now,
  $1.6ns$ for sequencing using Flip-Flops,
  $1.7ns$ for sequencing using Latches and
  $1.4ns$ for sequencing using Pulsed Latches.
- The implementation with flip-flops is faster than the one with latches in this case.

Using Domino Logic

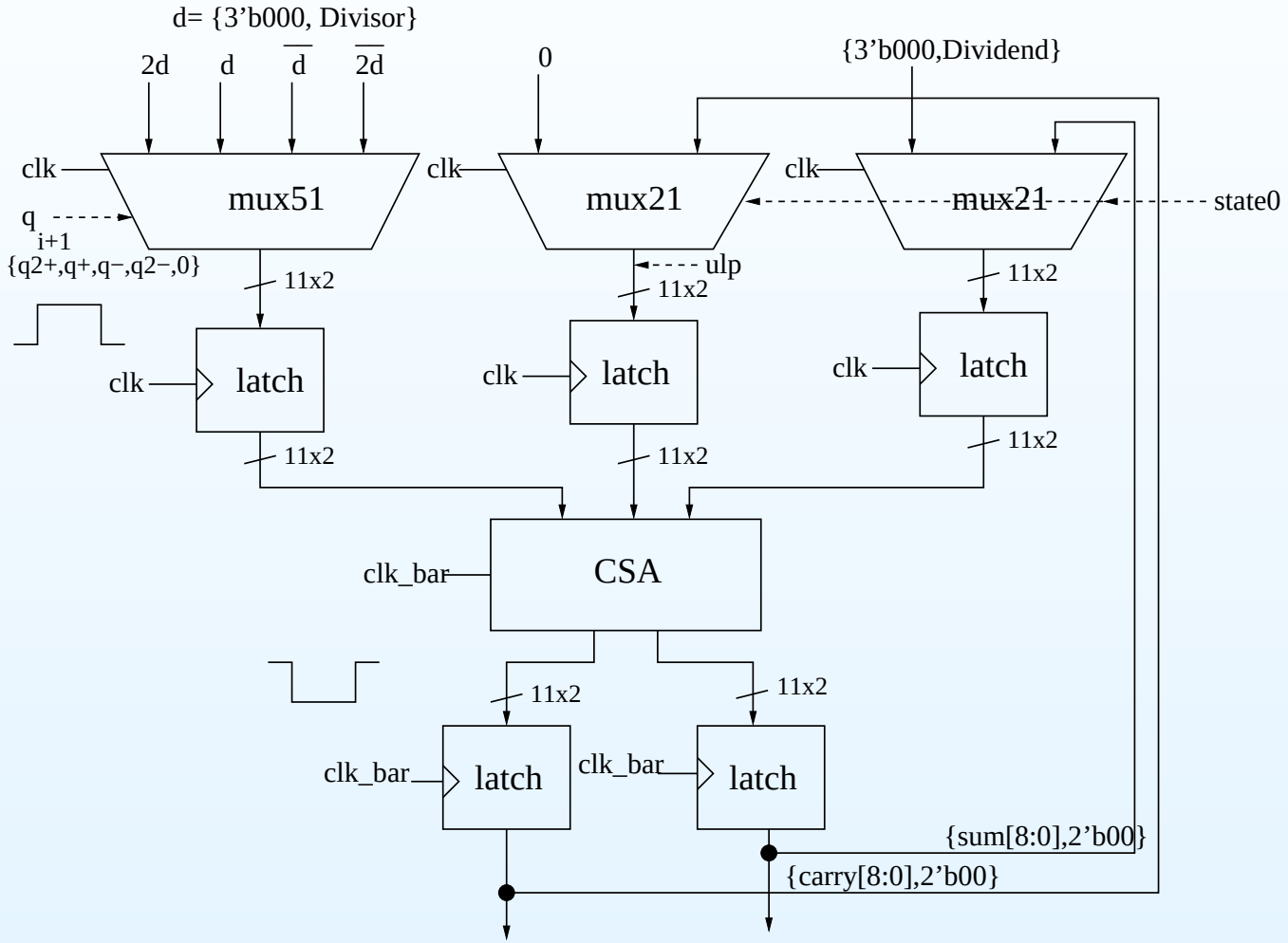
- Designers prefer domino logic for high-performance systems.
- Domino logic is considered twice as fast as the static CMOS logic, but this is often not the case.
- Requirement of dual-rail domino gates, remedies for charge-sharing problem and clocking techniques used limit the speed.
- Traditional domino clocking has large sequencing overhead.
- Overlapping clocks can be used to hide sequencing overheads, as shown ahead.

Traditional Domino Clocking



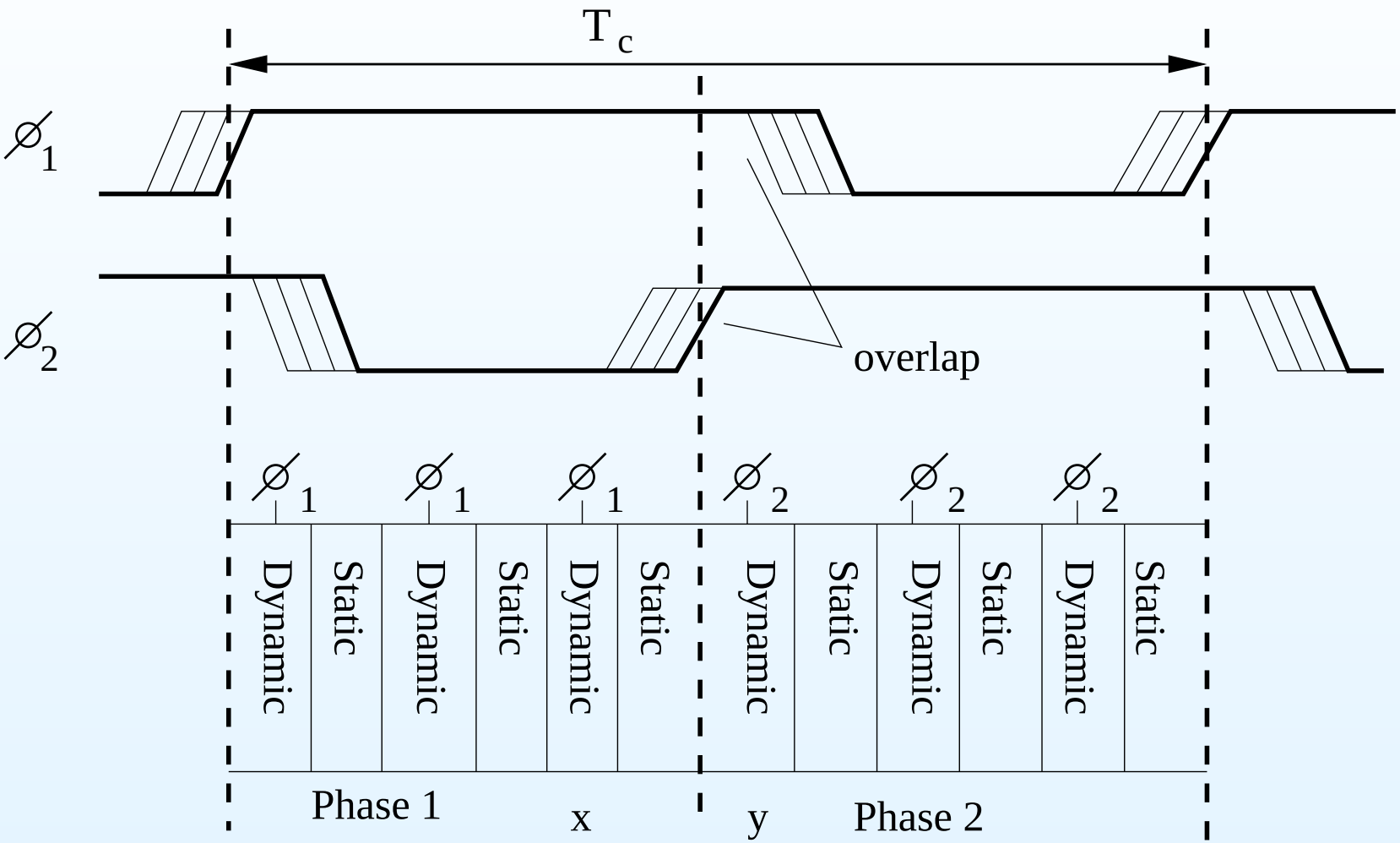
$$T_c = \Delta_{logic} + 2 \cdot \Delta_{DC} + 2 \cdot t_{skew}$$

Traditional Domino Clocking



$$T_c = \Delta_{logic} + 2 \cdot \Delta_{DC} + 2 \cdot t_{skew} = 1.6ns$$

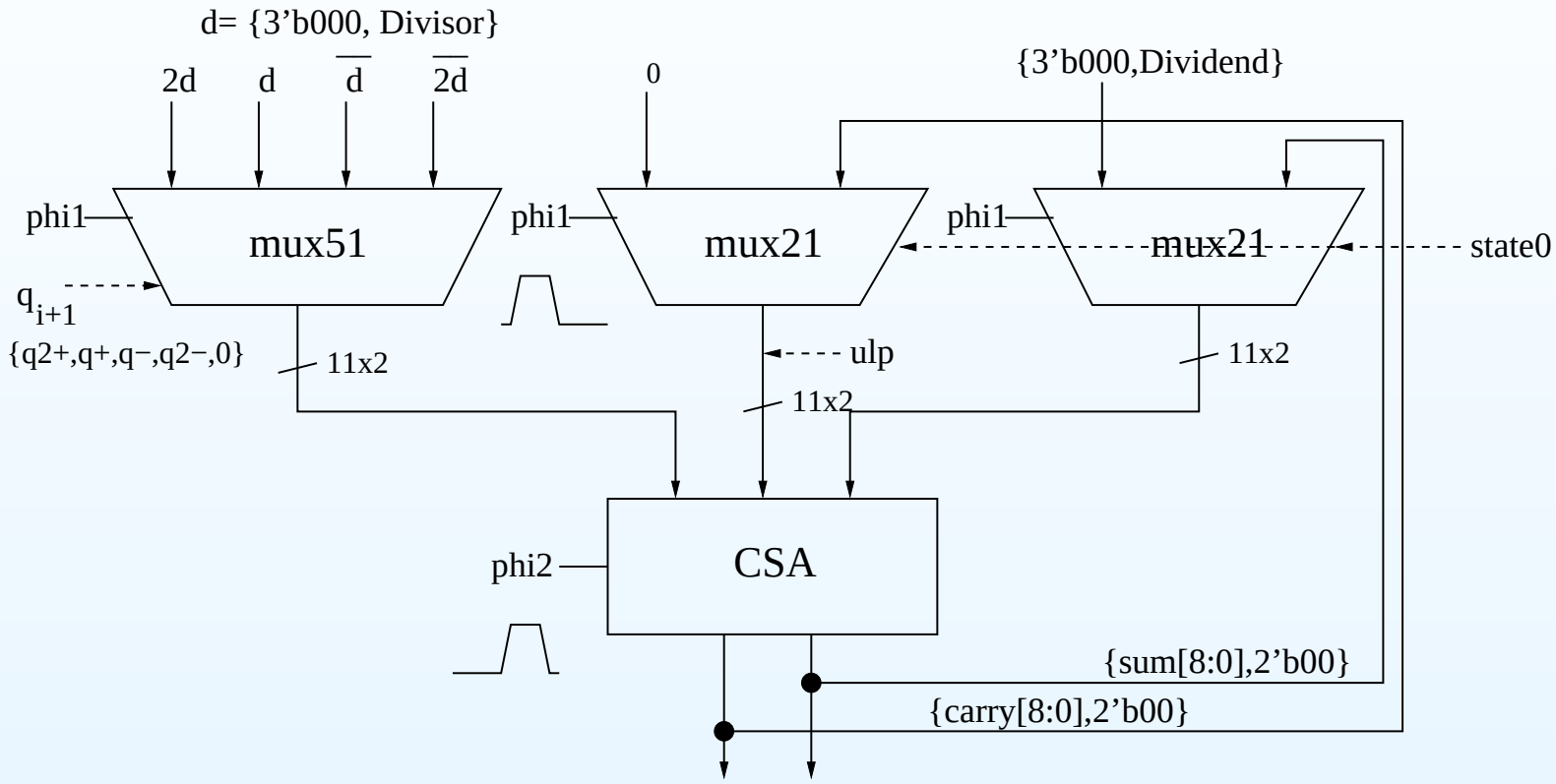
Skew Tolerant Domino Clocking



Constraints

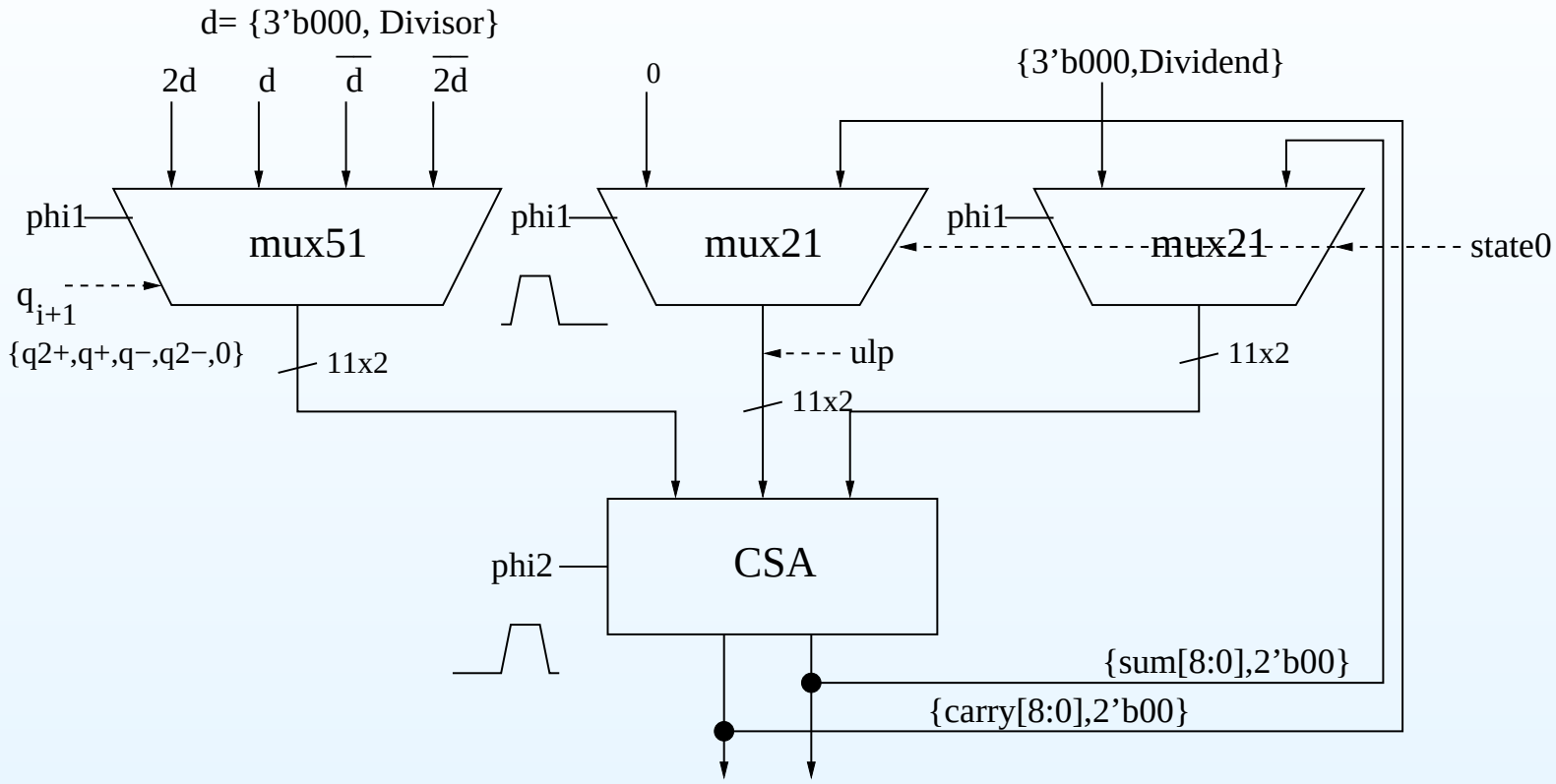
- $t_e = \frac{T_c}{N} + t_{hold} + t_{skew}$
- $t_p = t_{prech} + t_{skew}$
- $T_c = t_e + t_p$
- $t_{overlap} = \frac{N-1}{N} T_c - t_{prech} - t_{skew1} = t_{hold} + t_{skew2} + t_{borrow}$

Skew Tolerant Domino Implementation



$$T_c = \Delta_{logic}$$

Skew Tolerant Domino Implementation



$$T_c = \Delta_{logic} = 1.1ns$$

Comparison

Sequencing method	Time Period, $T_c ns(FO4)$	Number of devices (p-mos,n-mos)
Skew-tolerant domino	1.1 (6.2)	1052 (264, 788)
Pulsed Latches (xmux)	1.4 (7.9)	1096 (548, 548)
Pulsed Latches	1.6 (9)	964 (482, 482)
Flip-Flops (xmux)	1.6 (9)	1140 (570, 570)
Traditional domino	1.6 (9)	1932 (704, 1228)
Transparent Latches (xmux)	1.7 (9.6)	1360 (680, 680)
Transparent Latches	1.8 (10.1)	1228 (614, 614)
Flip-Flops	2 (11.2)	1008 (504, 504)

Conclusions

- Speed improves significantly with a better pipelining approach.
- Skew tolerant domino is the fastest implementation, yet has very few devices used.
- Alternatively, a pulsed latch implementation with transmission gates offers high clock frequency.
- Design complexity may also be an important factor while choosing a sequencing method.

Acknowledgements

- My deepest gratitude goes to my advisor Dr. James Stine.
- Inspiring working conditions at the Illinois Institute of Technology greatly facilitated my research.
- Thank you!